Q1. Write a program to implement Fuzzy Operations Union, Intersection, Complement, Algebraic sum, Algebraic product, Cartesian product.

```
#Accepting Fuzzy Set 1
n=int(input("Enter no. of elements of set 1: "))
d1={}
for i in range(n):
    keys=input()
    values=float(input())
    d1[keys]=values


#Accepting Fuzzy Set 2
n1=int(input("Enter no. of elements of set 2: "))
d2={}
for i in range(n1):
    keys1=input()
    values1=float(input())
    d2[keys1]=values1


#Fuzzy Operation Union
def union(A,B):
    result={}
    for i in A:
        if(A[i]>B[i]):
            result[i]=A[i]
        else:
            result[i]=B[i]
    print("Union of two sets is: ", result)


#Fuzzy Operation Intersection
def intersection(A,B):
    result={}
    for i in A:
        if(A[i]<B[i]):
            result[i]=A[i]
        else:
            result[i]=B[i]
    print("Intersection of two sets is: ", result)



#Fuzzy Operation Complement
def complement(A,B):
    result={}
    result1={}
    for i in A:
        result[i]=round(1-A[i],2)
    for i in B:
        result1[i]=round(1-B[i],2)
    print("Complement of set 1 is: ", result)
    print("Complement of set 2 is: ", result1)
```

```
#Fuzzy Operation Algebraic Sum
def algebrasum(A,B):
    result={}
    for i in A:
        result[i]=A[i]+B[i]-A[i]*B[i]
    print("Algebraic sum of two sets is", result) #round(result,2)


#Fuzzy Operation Algebraic Product
def algebraprod(A,B):
    result={}
    for i in A:
        result[i]=A[i]*B[i]
    print("Algebraic product of two sets is", result)
#round(result,2)


#Fuzzy Operation Cartesian Product
import numpy as np
def cartprod(A,B):
    R=[[] for i in range(len(A))]
    i=0
    for x in A:
        for y in B:
            R[i].append(min(A[x],B[y]))
        i +=1
    print("Cartesian product is \n", np.array(R),"\n")


print()
print("-----------------UNION---------------")
union(d1,d2)
print()
print("-------------INTERSECTION------------")
intersection(d1,d2)
print()
print("--------------COMPLEMENT-------------")
complement(d1,d2)
print()
print("------------ALGEBRAIC SUM------------")
algebrasum(d1,d2)
print()
print("---------ALGEBRAIC PRODUCT-----------")
algebraprod(d1,d2)
print()
print("----------CARTESION PRODUCT----------")
cartprod(d1,d2)
```

OUTPUT

```
Enter no. of elements of set 1: 4
x1
0.2
x2
0.3
x3
0.4
x4
0.5
Enter no. of elements of set 2: 4
x1
0.1
x2
0.2
x3
0.2
x4
1

-----------------UNION---------------
Union of two sets is:  {'x1': 0.2, 'x2': 0.3, 'x3': 0.4, 'x4':
1.0}

-------------INTERSECTION------------
Intersection of two sets is:  {'x1': 0.1, 'x2': 0.2, 'x3': 0.2,
'x4': 0.5}

--------------COMPLEMENT-------------
Complement of set 1 is:  {'x1': 0.8, 'x2': 0.7, 'x3': 0.6, 'x4':
0.5}
Complement of set 2 is:  {'x1': 0.9, 'x2': 0.8, 'x3': 0.8, 'x4':
0.0}

------------ALGEBRAIC SUM------------
Algebraic sum of two sets is {'x1': 0.28, 'x2': 0.44, 'x3': 0.52,
'x4': 1.0}

---------ALGEBRAIC PRODUCT-----------
Algebraic product of two sets is {'x1': 0.020000000000000004,
'x2': 0.06, 'x3': 0.08000000000000002, 'x4': 0.5}

----------CARTESION PRODUCT----------
Cartesian product is
 [[0.1 0.2 0.2 0.2]
 [0.1 0.2 0.2 0.3]
 [0.1 0.2 0.2 0.4]
 [0.1 0.2 0.2 0.5]]
```

Q2. Write a program to implement DE Morgan's Law.

```
n=int(input("Enter No. of elements for set1: "))
d1={}
for i in range(n):
    key=input()
    value=float(input())
    d1[key]=value
n=int(input("Enter No. of elements for set1: "))
d2={}
for i in range(n):
    key=input()
    value=float(input())
    d2[key]=value

def demorgans(A,B):
    result1={}
    result2={}
    first={}
    second={}
    for i in A:
        result1[i]=round(1-A[i],2)
    for i in B:
        result2[i]=round(1-B[i],2)
        first[i]=min(result1[i],result2[i])
        second[i]=max(result1[i],result2[i])
    print("Demorgan's First Law A'UB'=",first)
    print("Demorgan's Second Law A'nB'=",second)
demorgans(d1,d2)
```

OUTPUT

```
Enter No. of elements for set1: 4
x1
0.2
x2
0.3
x3
0.4
x4
0.5
Enter No. of elements for set1: 4
x1
0.1
x2
0.2
x3
0.2
x4
1
Demorgan's First Law A'UB'= {'x1': 0.8, 'x2': 0.7, 'x3': 0.6,
'x4': 0.0}
Demorgan's Second Law A'nB'= {'x1': 0.9, 'x2': 0.8, 'x3': 0.8,
'x4': 0.5}
```

Q3. Write a program to implement Max-Min Composition and Max-Product Composition.

i) Max-Min Composition

```
import numpy as np
def maxMin(x,y):
    z=[]
    for x1 in x:
        for y1 in y.T:
            z.append(max(np.minimum(x1,y1)))
    c=np.array(z).reshape((x.shape[0],y.shape[1]))
    print("R1oR2=> Max-Min Composition: \n",c)

r1=np.array([[0.4,0.6,0],[0.9,1,0.1]])
r2=np.array([[0.5,0.8],[0.1,1],[0,0.6]])
maxMin(r1,r2)
```

OUTPUT

```
R1oR2=> Max-Min Composition:
 [[0.4 0.6]
 [0.5 1. ]]
```

ii) Max-Product Composition

```
import numpy as np
def maxProduct(x,y):
    z=[]
    for x1 in x:
        for y1 in y.T:
            z.append(max(np.multiply(x1,y1)))
    c=np.array(z).reshape((x.shape[0],y.shape[1]))
    print("R1oR2=>Max-product Composition:\n",c)
r1=np.array([[0.4,0.6,0],[0.9,1,0.1]])
r2=np.array([[0.5,0.8],[0.1,1],[0,0.6]])
maxProduct(r1,r2)
```

OUTPUT

```
R1oR2=>Max-product Composition:
 [[0.2  0.6 ]
 [0.45 1.  ]]
```

Q4. Write a program to implement Lambda -Cut.

```
n=int(input("Enter no of elements of set 1:"))
d1={}
for i in range(n):
    keys=input()
    values=float(input())
    d1[keys]=values
n1=int(input("Enter no of elements of set 2:"))
d2={}
for i in range(n1):
    keys1=input()
    values1=float(input())
    d2[keys1]=values1
a=float(input("Enter the value for lambda cut for A set:"))
b=float(input("Enter the value for lambda cut for B set:"))
def cut(A,B):
    resultA={}
    resultB={}
    for i1 in A:
        if(A[i1]>=a):
            resultA[i1]=A[i1]
    print("Lambda cut for A is:",resultA.keys())
    for i2 in B:
        if(B[i2]>=b):
            resultB[i2]=B[i2]
    print("Lambda cut for B is:",resultB.keys())
cut(d1,d2)
```

OUTPUT

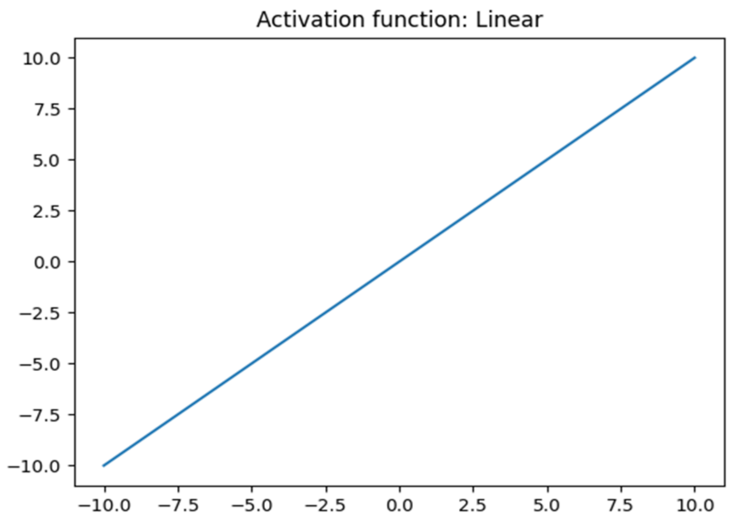
```
Enter no of elements of set 1:2
x1
0.2
x2
0.3
Enter no of elements of set 2:2
x1
0.9
x2
0.8
Enter the value for lambda cut for A set:0.3
Enter the value for lambda cut for B set:0.7
Lambda cut for A is: dict_keys(['x2'])
Lambda cut for B is: dict_keys(['x1', 'x2'])
```

Q5. Write a program to implement Activation Funtion, Linear/Identity, Binary Step, Sigmoid, ReLu activation function.

i) Activation Function Linear/Identity

```
import matplotlib.pyplot as plt
import numpy as np
def linear(x):
    return x

x=np.linspace(-10,10)
plt.plot(x,linear(x))
plt.title('Activation function: Linear')
plt.show()
```

OUTPUT

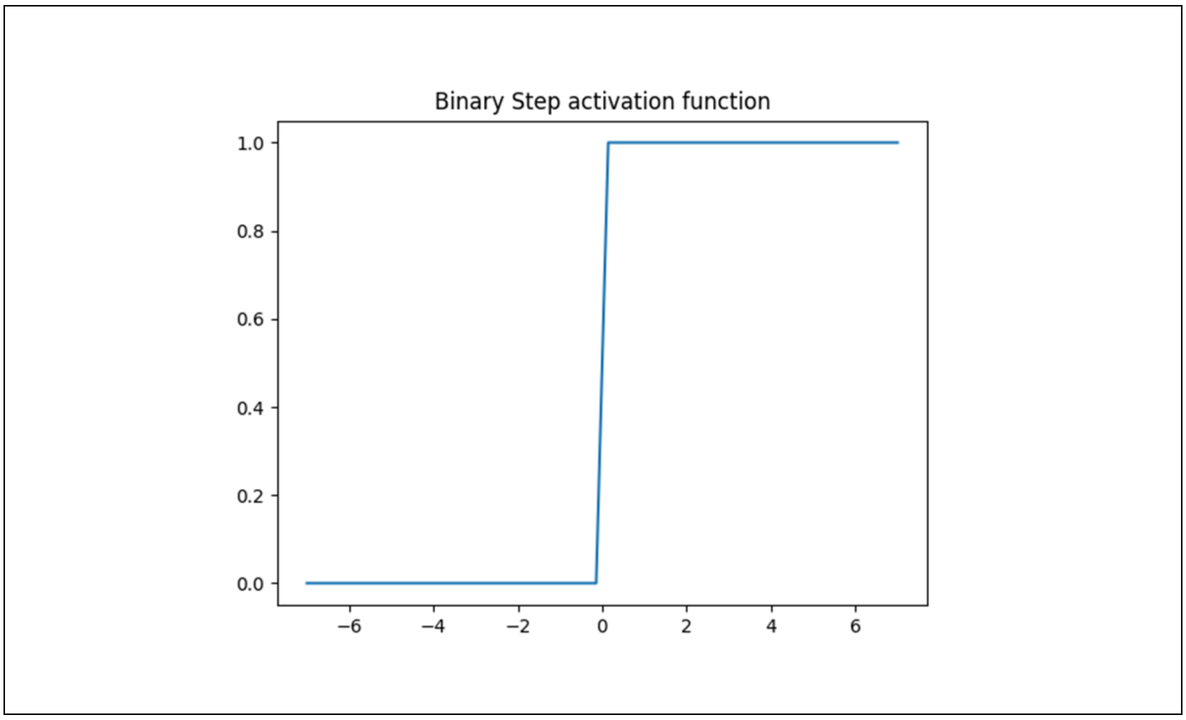


ii) Binary Step

```
import numpy as np
import matplotlib.pyplot as plt
def binarystep(x):
    lst=[]
    for i in x:
        if i >=0:
            lst.append(1)
        else:
            lst.append(0)
    return lst
```

```
arr=np.linspace(-7,7)
plt.plot(arr, binarystep(arr))
plt.title('Binary Step activation function')
plt.show()
```

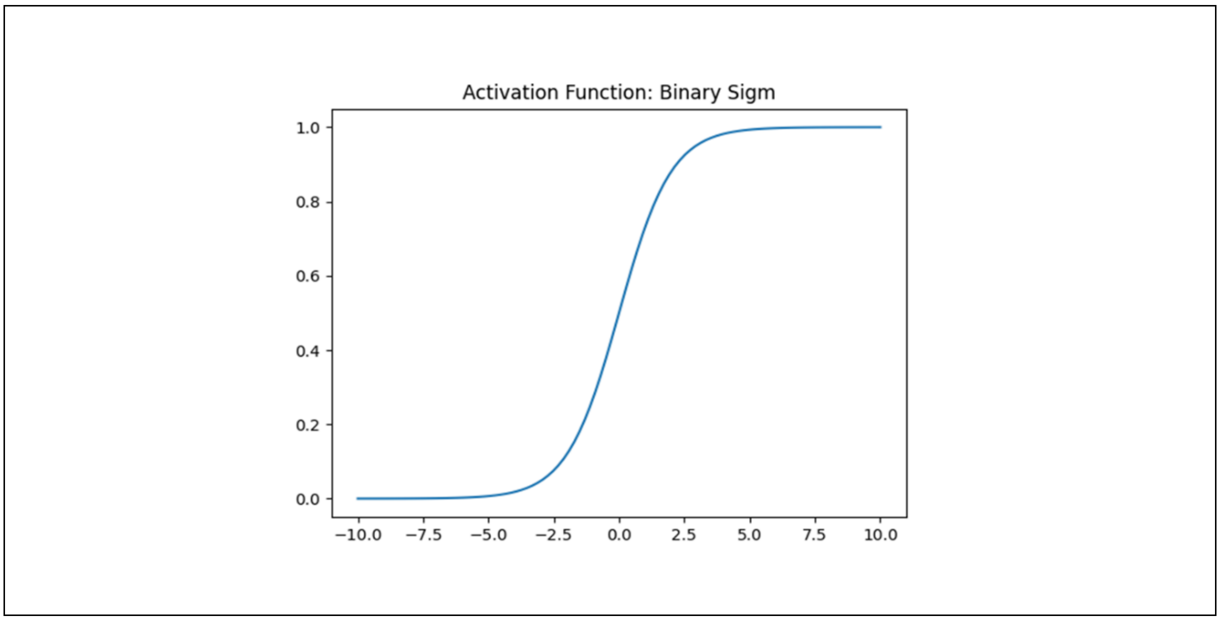
OUTPUT



iii) Sigmoidal Activation Function
a) Binary Sigmoidal Activation Function

```
import numpy as np
x=np.linspace(-10,10,100)
def sigmoid(x):
    return 1/(1+np.exp(-x))
from matplotlib import pyplot as plt
plt.plot(x,sigmoid(x))
plt.title('Activation Function: Binary Sigm')
plt.show()
```

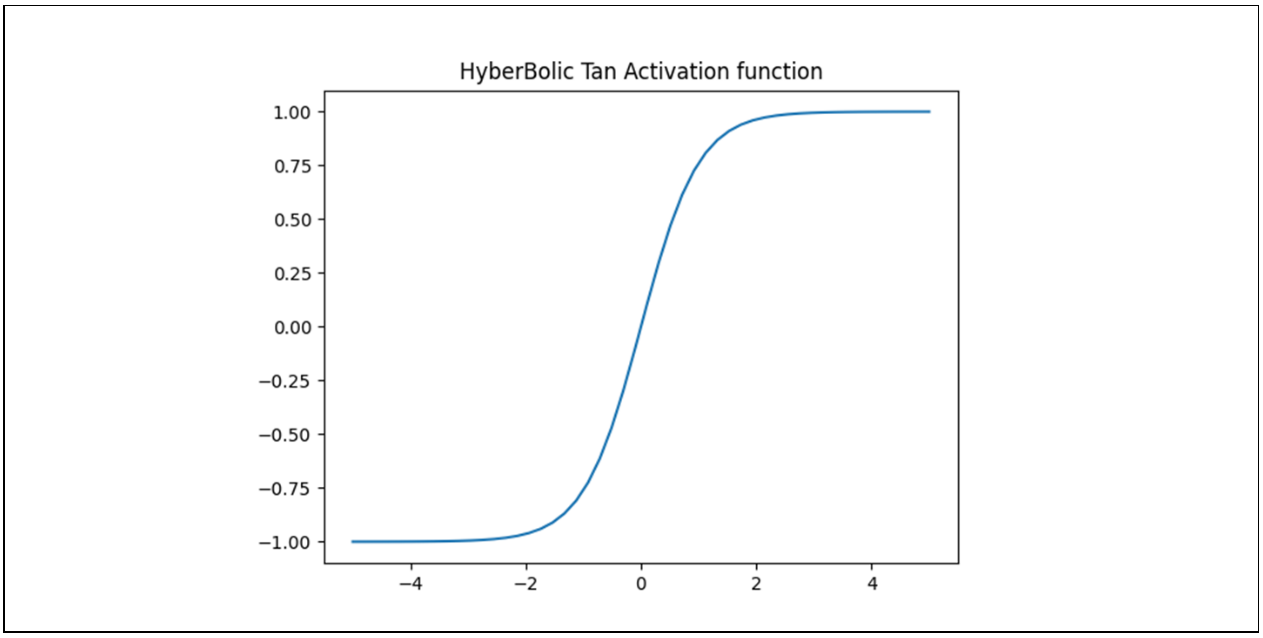
OUTPUT



b) Bipolar Sigmoidal Activation Function

```
import numpy as np
import matplotlib.pyplot as plt

def Hyperbolictan(t):
    return np.tanh(t)

t=np.linspace(-5,5)
plt.plot(t,Hyperbolictan(t))
plt.title('HyberBolic Tan Activation function')
plt.show()
```

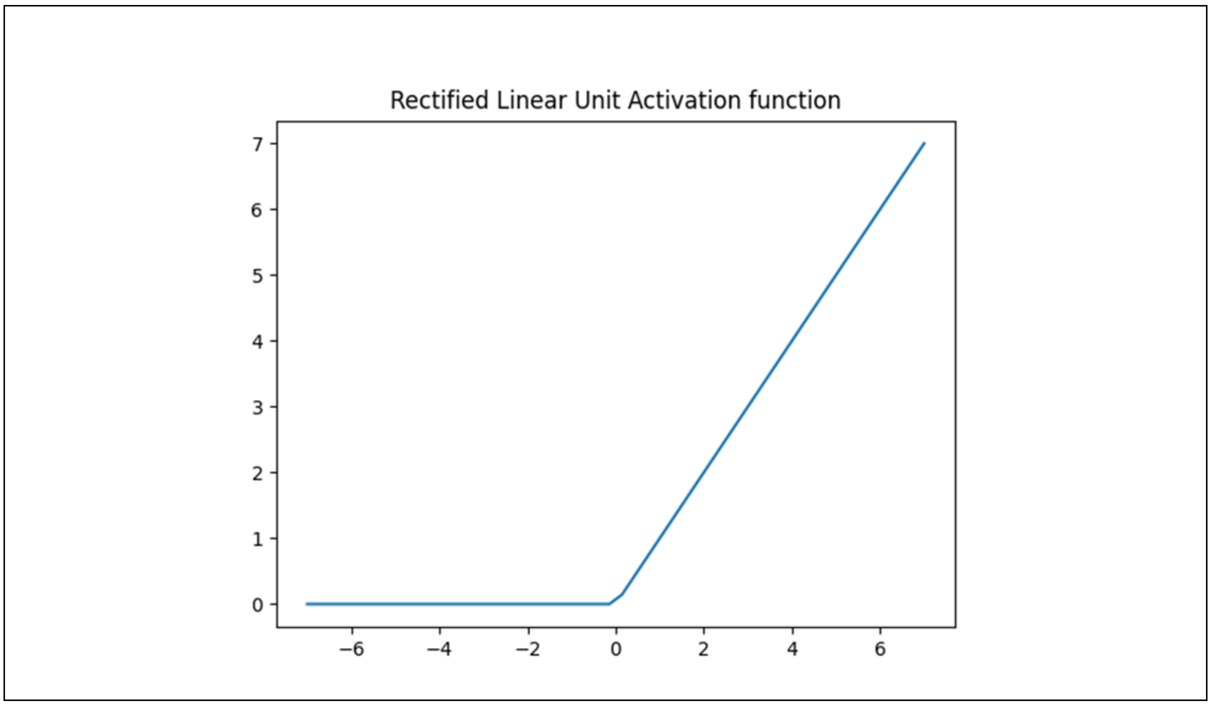
OUTPUT



iv) ReLu Activation Function

```
import numpy as np
import matplotlib.pyplot as plt

def RectifiedLinearUnit(x):
    lst=[]
    for i in x:
        if i >=0:
            lst.append(i)
        else:
            lst.append(0)
    return lst

arr=np.linspace(-7,7)
plt.plot(arr, RectifiedLinearUnit(arr))
plt.title('Rectified Linear Unit Activation function')
plt.show()
```

OUTPUT



Q6. Write a program to implement Perceptron Learning Rule.

```
import numpy as np
features=np.array([[0,0],[0,1],[1,0],[1,1]])
labels=np.array([0,0,0,1])
w=[0.9,0.9]
threshold=0.5
learning_rate=0.1
epoch=20
for j in range(0,epoch):
    print("epoch",j)
    newdelta=0
    for i in range(0,features.shape[0]):
        actual=labels[i]
        instance=features[i]
        x0=instance[0]
        x1=instance[1]
        sum_unit=x0*w[0] + x1 *w[1]
        if sum_unit>threshold:
            result=1
        else:
            result=0

        delta=actual-result
        newdelta=newdelta+abs(delta)

        print("Prediction:",result,"whereas actual was",
actual,"(error:",delta,")")
        w[0]=w[0]+delta * learning_rate
        w[1]=w[1]+delta * learning_rate
    print("-----------------------------------------------")
    if newdelta==0:
        break
print(w)
```

OUTPUT

```
epoch 0
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 1 whereas actual was 0 (error: -1 )
Prediction: 1 whereas actual was 0 (error: -1 )
Prediction: 1 whereas actual was 1 (error: 0 )
-----------------------------------------------
epoch 1
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 1 whereas actual was 0 (error: -1 )
Prediction: 1 whereas actual was 0 (error: -1 )
Prediction: 1 whereas actual was 1 (error: 0 )
-----------------------------------------------
epoch 2
Prediction: 0 whereas actual was 0 (error: 0 )
```

```
Prediction: 1 whereas actual was 0 (error: -1 )
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 1 whereas actual was 1 (error: 0 )
------------------------------------------------
epoch 3
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 0 whereas actual was 0 (error: 0 )
Prediction: 1 whereas actual was 1 (error: 0 )
------------------------------------------------
[0.40000000000000013, 0.40000000000000013]
```

Q7. Write a program to implement Feed Forward Network.

```
import numpy as np
def relu(n):
    if n<0:
        return 0
    else:
        return n

inp=np.array([[-1,2],[2,2],[3,3]])
weights=[np.array([3,3]),np.array([1,5]),np.array([3,3]),np.ar
ray([1,5]),np.array([2,-1])]
for x in inp:
    node0=relu((x*weights[0]).sum())
    node1=relu((x*weights[1]).sum())
    node2=relu(([node0,node1]*weights[2]).sum())
    node3=relu(([node0,node1]*weights[3]).sum())
    op=relu(([node2,node3]*weights[4]).sum())
    print("Inputs:",x,"output:",op)
```

OUTPUT

```
Inputs: [-1  2] output: 24
Inputs: [2 2] output: 72
Inputs: [3 3] output: 108
```

Q8. Write a program for implementing the back Propagation Algorithm.

```
import numpy as np
def sigmoid(x, deriv=False):
    if(deriv==True):
        return x*(1-x)
    return 1/(1+np.exp(-x))

X=np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
Y=np.array([[0,0,1,1]]).T
np.random.seed(1)
sy=2*np.random.random((3,1))-1

for i in range(1000):
    layer0=X
    layer1=sigmoid(np.dot(layer0,sy))
    error=Y-layer1
    layerdelta=error*sigmoid(layer1,True)

    sy+=np.dot(layer0.T,layerdelta)

print("Output after training:")
print(layer1)
print("Actual output")
print(Y)
```

OUTPUT

```
Output after training:
[[0.03178421]
 [0.02576499]
 [0.97906682]
 [0.97414645]]
Actual output
[[0]
 [0]
 [1]
 [1]]
```

Q9. Write a program for solving Linearly Separable Problem using Perceptron Model.

```
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))
def sigmoid_der(x):
    return x *(1-x)
class NN:
    def __init__(self,inputs):
        self.inputs=inputs
        self.l=len(self.inputs)
        self.l1=len(self.inputs[0])
        self.wi=np.random.random((self.l1, self.l))
        self.wh=np.random.random((self.l,1))

    def think(self,inp):
        s1=sigmoid(np.dot(inp,self.wi))
        s2=sigmoid(np.dot(s1,self.wh))
        return s2

    def train(self, inputs, outputs, it):
        for i in range(it):
            l0=inputs
            l1=sigmoid(np.dot(l0,self.wi))
            l2=sigmoid(np.dot(l1,self.wh))
            l2_err=outputs-l2
            l2_delta=np.multiply(l2_err,sigmoid_der(l2))
            l1_err=np.dot(l2_delta, self.wh.T)
            l1_delta=np.multiply(l1_err,sigmoid_der(l1))

            #updates the weights
            self.wh+=np.dot(l1.T, l2_delta)
            self.wi+=np.dot(l0.T, l1_delta)

inputs=np.array([[0,0],[0,1],[1,0],[1,1]])
outputs=np.array([[0],[1],[1],[0]])

n=NN(inputs)
print("Before training")
print(n.think(inputs))
n.train(inputs,outputs,10000)
print("After training")
print(n.think(inputs))
```

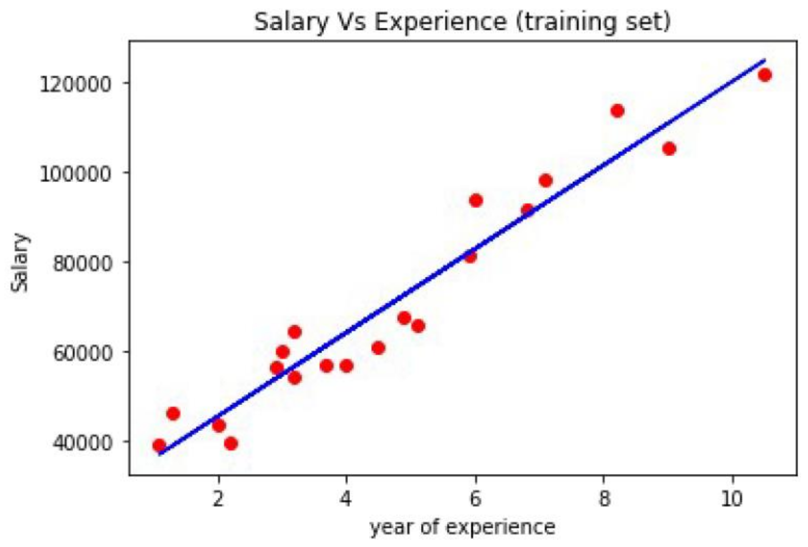
OUTPUT

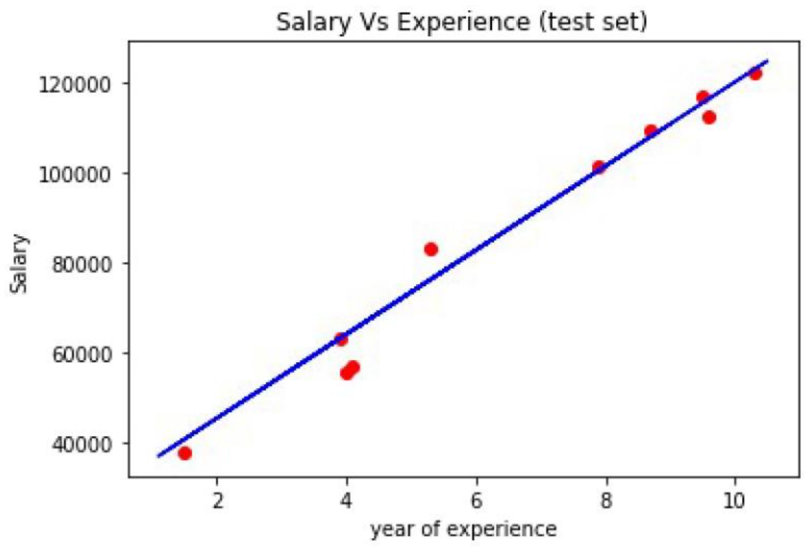
```
Before training
[[0.69254302]
 [0.73636244]
 [0.73247292]
 [0.76901515]]
After training
[[0.01929362]
 [0.98353545]
 [0.98169802]
 [0.01641891]]
```

Q10. Write a program to develop Supervised Learning Algorithm (Linear Regression).

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset=pd.read_csv('/home/mcs16/Desktop/salary_data.csv')
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:,1].values

from sklearn.model_selection import train_test_split
X_train, X_test,
y_train,y_test=train_test_split(X,y,test_size=1/3,
random_state=0)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred=regressor.predict(X_test)

plt.scatter(X_train, y_train, color='red')
plt.plot(X_train,regressor.predict(X_train), color='blue')
plt.title('Salary Vs Experience (training set)')
plt.xlabel('year of experience')
plt.ylabel('Salary')
plt.show()

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train,regressor.predict(X_train), color='blue')
plt.title('Salary Vs Experience (test set)')
plt.xlabel('year of experience')
plt.ylabel('Salary')
plt.show()
```

OUTPUT


Salary Vs Experience (training set)
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
year of experience


Salary Vs Experience (test set)
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
year of experience

Q11. Write a program to implement Crossover Operation of Genetic Algorithm.

```
import random
def crossover(parent1,parent2,point):
    p1,p2=list(parent1), list(parent2)
    for i in range(point,len(p1)):
        p1[i],p2[i]=p2[i],p1[i]
    p1,p2=''.join(p1),''.join(p2)
    return p1,p2

parent1='1011000101'
parent2='0100100101'

print('parent1:',parent1)
print('parent2:',parent2)
point=4

print('Crossover point:',point)
offspring1,offspring2=crossover(parent1,parent2,point)

print('offspring1:',offspring1)
print('offspring2:',offspring2)
```

OUTPUT

```
parent1: 1011000101
parent2: 0100100101
Crossover point: 4
offspring1: 1011100101
offspring2: 0100000101
```